Design Abstractions for Context-Awareness

José Luiz Fiadeiro

University of Leicester

joint work with
Antónia Lopes
Architectures for Mobility

ATX Software SA
ISTI-CNR
University of Florence
University of Leicester
University of Lisbon
University of Munich
University of Pisa

IST-2001-32747
Architectures for Mobility
Jan 02 – Apr 05
Motivation

- System operation depends on **environmental** resources
  
<table>
<thead>
<tr>
<th>Network connectivity</th>
<th>CPU</th>
<th>Directory Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>Memory</td>
<td>Printers</td>
</tr>
<tr>
<td>Battery Power</td>
<td>Screen size</td>
<td></td>
</tr>
</tbody>
</table>

- That are subject to **change**
  
  - **Mobility** of components / hosts
  - **Fragility** of communication
  - **Variety** of devices
  - ...

- Yet contained in an “architectural” structure
Motivation

Dealing with the **Dynamics of Environments**

- **Traditional approach**: based on **Exceptions**.
  Systems are developed for being executed using particular resources; at runtime, changes on operating conditions are not anticipated and raise “exceptions”.

- **Context-Aware Computing Paradigm**: based on a notion of **Context**.
  Systems have means to observe the surrounding environment and are developed taking into account different conditions in which they can be required to operate.
The challenge

Formalisms for designing ubiquitous systems need to
- address contexts as first-class concerns
- support “personalised” contexts
- relate context awareness with architectural dimensions

which requires
- the identification of essential features of contexts
- design primitives for defining contexts
- abstractions for modelling context-awareness
Separation of concerns

**Mobility** as a separate concern

Compositionality wrt refinement
wrt evolution
Example

• In a **sender-receiver system**, a sender produces, in one go, words of N bits that are then transmitted, one by one, to a receiver through **synchronous message passing**.

• The sender is fixed and the receiver is a mobile component: once a word defining a location is received, the component should move its execution to that location; if this is not possible, it discards that word and starts receiving another one.
Context awareness

- Computations, as performed by individual components, are constrained by the resources and services available at the positions where the components are located.

- Communication among components can only take place when they are located in positions that are in touch with each other.

- Movement of components from one position to another is constrained by reachability.

  a piece of mobile code that relies on numerical operations with high precision will fail to compute when executing in locations where memory is scarce either because the physical links that support comm between the positions of the space of mobility may be subject to failures or interruptions, making communication temporarily impossible.

  typically the space has some structure given by walls and doors or barriers erected in communication networks by system administrators.
Mathematically speaking...

- Systems designed in terms of actions, communication channels and context observables
- Architectural aspects formalised over categories of designs (superposition and refinement as morphisms)
- Behavioural aspects formalised algebraically:
  - Algebraic specification of the data that can be transmitted and the operations that perform the computations that are required
  - Space of mobility represented by a distinguished data type \( \text{Loc} \)
  - Context observables defined over the algebra

\[
\begin{align*}
  \text{rssv}: & \rightarrow \text{nat}_\infty x 2^\Omega \text{ the resources and services that are available for computation} \\
  \text{bt}: & \rightarrow 2^{\text{Loc}} \text{ how locations are in-touch through communication} \\
  \text{reach}: & \rightarrow 2^{\text{Loc}} \text{ how locations can be reached through movement}
\end{align*}
\]
Example. Designing an airport luggage handling system

design located cart is
in loc pos:Loc
in next:Loc
prv busy@pos:bool, dest@pos:Loc
do move@pos[]: ¬busy∧pos≠dest,false → true
[] dock@pos[busy]: ¬busy∧pos=dest,false → busy'
[] undock@pos[busy]:
  busy∧pos=dest,false → ¬busy'||dest'=next
Example. Controlling how a cart moves

design controlled located cart is
outloc pos:Loc
inloc cp:Loc
in next:Loc
prv busy@pos:bool, dest@pos:Loc,
in@cp:bool, mode@pos:[slow,fast]
do move@pos: ¬busy ∧ pos ≠ dest, false → c(pos, pos', mode)
[] dock@pos: ¬busy ∧ pos = dest, false → busy'
[] undock@pos: busy ∧ pos = dest, false → ¬busy' || dest' = next
[] prv enter @pos: true → mode' = slow
    @cp: ¬in → in'
[] prv leave @pos: true → mode' = fast
    @cp: in → ¬in'

CommUnity with Distribution: Example
Externalisation of the distribution aspects

design controlled located cart is
    outloc pos:Loc
    inloc cpoint:Loc
    in  next:Loc
prv busy@pos:boolean, dest@pos:Loc,
    in@cpoint:boolean, mode@pos:[slow,fast]
do move@pos: ¬busy
    ∧ pos ≠ dest
    false → c(pos,pos',mode)
    dock@pos: ¬busy
    ∧ pos=dest
    false → busy'
    undock@pos: busy
    ∧ pos=dest
    false → ¬busy'
    | |
    dest'=next
prv enter @pos: true → mode'=slow
    @cpoint: ¬in
    → in'
prv leave @pos: true → mode'=fast
    @cpoint: in
    → ¬in'

design mode controller is
    outloc theirs:Loc
    inloc mine:Loc
    in  next:Loc
prv in@cpoint:boolean, mode@pos:[slow,fast]
do control@theirs: true → c(theirs,theirs',mode)
    prv enter @theirs: true → mode'=slow
    @mine: ¬in → in'
    prv leave @theirs : true → mode'=fast
    @mine: in → ¬in'
Categorical semantics

design located cart is

...  

design mode controller is

...  

design controlled located cart is

outloc pos:Loc
inloc cpoint:Loc
in next:Loc
prv busy@pos:bool, dest@pos:Loc, in@cpoint:bool, mode@pos:[slow,fast]
do move@pos: ¬busy∧pos≠dest,false → c(pos,pos′,mode)
   • dock@pos: ¬busy∧pos=dest,false → busy'
   • undock@pos: busy∧pos=dest,false → ¬busy'||dest'=next
   • prv enter @pos: true → mode'=slow @cpoint: ¬in → in'
   • prv leave @pos: true → mode'=fast @cpoint: in → ¬in'
Separation of concerns

design fixed is
out loc stay:Loc
out count@where:nat
do inc@where: true \rightarrow count' = count + 1
[] reset@where: true \rightarrow count' = 0
Conclusions

Our architectural approach supports

- an incremental development (associativity) of location-aware systems; this makes it easier to cope with increased complexity and promotes reuse

- principled-ways of making models location-aware; in this process designers can be assisted through libraries with location and distribution connectors modelling standard solutions
www.fiadeiro.org/jose/CommUnity